2012

SequenceL Regular Expression Library

Documentation

This document describes the functions and data structures of the SequenceL Regular Expression Library.

Bryant Nelson

7/9/2012

Contents

Regular Expression Syntax/Semantics

Ordinary Characters

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklmnopqrstuvwxyz"
"0123456789"
";<=>@_`{}~"
"!#%&',-/:"
```

- Space Character: `" "`
- Tab Character: `"\t"`
- New-Line Character: `"\n"`
- Double Quote Character: `"`

Special Characters

- Backslash: `'\'`
- Period: `'.'`
- Square Brackets: `'[' ']'`
- Caret: `'^'`
- Dollar Sign: `'$'`
- Asterisk: `'*'`
- Plus Sign: `'+'`
- Question Mark: `'?'`
- Parentheses: `'(' ')'`
- Pipe: `'|'`

Bracketed Character Sets

Positive Character Set

- A list of characters enclosed within Square Brackets is referred to as a Positive Character Set as long as the first character in that list is not a Caret.
- A Positive Character Set contains all characters in the list of characters.
- Example:

  ```
  "[abc]"
  ```

  Defines the Bracketed Character Set containing the characters ‘a’, ‘b’, and ‘c’.

Negative Character Set

- A list of characters enclosed within Square Brackets is referred to as a Negative Character Set as long as the first character in that list is a Caret.
- A Negative Character Set contains all characters in the Alphabet which are NOT in the list of characters.
- The first Caret is not part of the list of characters. It is used only to specify a Negative Character Set.
- Example:

  ```
  "[^abc]"
  ```

  Defines the Bracketed Character Set containing every character EXCEPT ‘a’, ‘b’, and ‘c’.

Built-In Character Sets

RegEx String	Description
\d	Contains all Numeric Characters. [0123456789]
\D	The negation of \d. Contains all Non-Numeric Characters.
\a	Contains all Upper and Lower Case Letter Characters. [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]
\A	The negation of \a. Contains all Non-Letter Characters.
\w	Contains all “Word” Characters. [0123456789abcdefghijklmnopqrstuvqxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_]
\W	The negation of \w. Contains all Non-“Word” Characters.
\s	Contains all Whitespace Characters. [Space, Tab, New-Line]
\S	The negation of \a. Contains all Non-Whitespace Characters.

Matching

A Regular Expression is a String of Ordinary and/or Special Characters which can Match another String based on the following rules.

- The Regular Expression containing only the Ordinary Character ***C*** matches the string of length one containing only the character ***C***.
- The Regular Expression containing only the Special Character ***C*** preceded by a Backslash, matches the string of length one containing only the character ***C***.
- The Regular Expression containing only the Bracketed Character Set ***B*** matches the string of length one containing only the character ***C***, where ***C*** is a member of ***B***.
- The Regular Expression containing only the Period, or Wildcard Character, matches any string of length one.
- The Regular Expression "(***X***)", where ***X*** is a Regular Expression, matches all strings matched by ***X***.
- The Regular Expression "***X***|***Y***", where ***X*** and ***Y*** are Regular Expressions, matches the string ***S,*** if either ***X*** matches ***S*** or ***Y*** matches ***S***.
- The Regular Expression "***XY***", where ***X*** and ***Y*** are Regular Expressions, matches the string ***S,*** of length ***N***, if there exists some integer ***I*** such that ***X*** matches ***S[1 … I]*** and ***Y*** matches ***S[I+1 … N]***.
- The Regular Expression "***X*** *", where ***X*** is a Regular Expression, matches the string ***S*** if ***S*** is composed only of one or more concatenated strings that are matched by ***X***, or ***S*** is the empty string.
- The Regular Expression "***X*** ?" , where ***X*** is a Regular Expression, matches the string ***S*** if ***X*** matches ***S*** or ***S*** is the empty string.
- The Regular Expression "***X*** +", where ***X*** is a Regular Expression, matches the string ***S*** if ***S*** is composed only of one or more concatenated strings that are matched by ***X***.
- The Regular Expression "^ ***X***", where ***X*** is a Regular Expression, matches the substring ***T*** of string ***S*** if ***T*** occurs at the beginning of ***S***.
- The Regular Expression "***X*** $", where ***X*** is a Regular Expression, matches the substring ***T*** of string ***S*** if ***T*** occurs at the end of ***S***.

Operation Precedence

1. Character Escaping `'\'`
2. Bracketed Character Classes `'[' ']'`
3. Parenthetical Sub-Expressions `'(' ')'`
4. Repetition Operators `'*' '+' '?'`
5. Concatenation
6. Union `'|'`
7. Beginning and Ending Anchoring `'^' '$'`

Library Functions

Matches(String S, String E)

Description

Inputs a String ***S*** and a Regular Expression String ***E*** and determines whether ***E*** Matches ***S*** as defined above.

For Example:

```
Matches("ca", "(b|c)a?")
```

Returns:

```
true
```

FirstMatch(String S, String E)

Description

Inputs a String ***S*** and a Regular Expression String ***E*** and returns a Match Struct ***M***, (Flag, Begin, End) where Flag is a Boolean, and Begin and End are Integers.

If ***M.Flag*** is True, then ***M.Begin*** and ***M.End*** are the indices in ***S*** of the first and last characters of the first substring of ***S*** string that matches ***E***. If ***M.Flag*** is False there are no substrings of ***S*** Matched by ***E***.

For Example:

```
FirstMatch("cabcr", "ab");
```

Returns:

```
(Flag: true, Begin: 2, End: 3)
```

ReplaceFirst(String S_1, String E, String S_2)

Description

If ***FirstMatch(S_1, E).Flag*** is true, this returns the String obtained from ***S_1*** by replacing the characters between positions ***FirstMatch(S_1, E).Begin*** and ***FirstMatch(S_1, E).End*** with the String ***S_2***.
Otherwise it returns ***S_1***.

For Example:

```
ReplaceFirst("cabcabr", "ab", "xyz")
```

Returns:

```
"cxyzcabr"
```

ReplaceAll(String S_1, String E, String S_2)

Description

If ***FirstMatch(S_1, E)..Flag*** is true, this returns the String obtained from concatenating the characters between positions 1 and ***FirstMatch(S_1, E).Begin***, ***S_2***, and ***ReplaceAll(S_1, E, S_3)***, where ***S_3*** is the substring of ***S_1*** between ***FirstMatch(S_1, E).End*** and the size of ***S_1.***
Otherwise it returns ***S_1***.

For Example:

```
ReplaceAll("Hello World", "(el+o)|(orld)", "ey")
```

Returns:

```
"Hey Wey"
```

FindFirst(String S, String E)

Description

If ***FirstMatch(S, E).Flag*** is true, this returns the SubString of ***S*** between ***FirstMatch(S, E).Begin*** and ***FirstMatch(S, E).End***.
Otherwise it returns “”.

For Example:

```
FindFirst("Total: 123 Pounds", "\\d+")
```

Returns:

```
"123"
```

FindAll(String S, String E)

Description

If ***FirstMatch(S, E).Flag*** is true, this returns the list obtained from concatenating the singleton list containing the SubString of ***S*** between ***FirstMatch(S, E).Begin*** and ***FirstMatch(S, E).End***, and ***FindAll(T, E)***, where ***T*** is the substring of ***S*** between ***FirstMatch(S, E).End*** and the size of ***S.***
Otherwise it returns ***[]***.

For Example:

```
FindAll("The fox jumped over.", "\\a+")
```

Returns:

```
["The","fox","jumped","over"]
```

Quick Reference Guide

Match Struct:

Member Name	Member Type	Member Description
Flag	Boolean	True when a valid match is found. False otherwise.
Begin	Integer	The starting index of a match if Flag is true. -1 otherwise.
End	Integer	The ending index of a match if Flag is true. -1 otherwise.

Matches(String S, String E)

Type	Description
Inputs	
String ***S***	A String to match against the Regular Expression
Regular Expression String ***E***	The Regular Expression to match against.
Output	
Boolean	True if the Regular Expression accepts the String. False otherwise.

FirstMatch(String S, String E)

Type	Description
Inputs	
String	A String to search for a substring that matches the Regular Expression Struct.
Regular Expression String ***E***	The Regular Expression to match against.
Output	
Match Struct	A Struct representing the results of the search.

ReplaceFirst(String S_1, String E, String S_2)

Type	Description
Inputs	
String $\boldsymbol{S_1}$	A String that will have the first occurrence of ***E*** replaced by $\boldsymbol{S_2}$.
Regular Expression String ***E***	The Regular Expression to match against.
String $\boldsymbol{S_2}$	The new String used to replace.
Output	
String	The string obtained from $\boldsymbol{S_1}$ by replacing the first occurring substring that is accepted by ***E*** with $\boldsymbol{S_2}$.

ReplaceAll(String S_1, String E, String S_2)

Type	Description
Inputs	
String ***S_1***	A String that will have all distinct occurrences of ***E*** replaced by ***S_2***.
Regular Expression String ***E***	The Regular Expression to match against.
String ***S_2***	The new String used to replace.
Output	
String	The string obtained from ***S_1*** by replacing all distinctly occurring substrings that are accepted by ***E*** with ***S_2***.

FindFirst(String S, String E)

Type	Description
Inputs	
String ***S***	A String that will be searched.
Regular Expression String ***E***	The Regular Expression to match against.
Output	
String	The first substring of ***S*** that is matched by ***E***.

FindAll(String S, String E)

Type	Description
Inputs	
String ***S***	A String that will be searched.
Regular Expression String ***E***	The Regular Expression to match against.
Output	
List<String>	The list containing all distinct substrings of ***S*** that are matched by ***S***.